Combining Automated Theorem Provers with Symbolic Algebraic Systems — Position Paper —

Johann Schumann

Automated Reasoning,
Institut für Informatik, TU München,
and
Automated Software Engineering
NASA Ames
email: `schumann@ptolemy.arc.nasa.gov`

Postal address:
Johann Schumann
2680 Fayette Dr. # 202
Mountain View, Ca. 94040
USA

1 Introduction

I personally became interested in ATPs and symbolic algebraic systems when I got MuMath/MuSimp running on a 48K(!) TRS80 back in '82. I even implemented a small propositional theorem prover (a variant of Wang's algorithm [Wan60]) on top of it.

In contrast to pure mathematical applications where automated theorem provers (ATPs) are quite capable, proof tasks arising from real-world applications from the area of Software Engineering show quite different characteristics: they usually do not only contain much arithmetic (albeit often quite simple one), but they also often contain reasoning about specific structures (e.g., graphs, sets,...). Furthermore, particularly in embedded applications where the proof tasks are generated automatically proof obligations usually contain many redundant and unoptimized parts. Thus, an ATP must be capable of performing *reasoning* together with a fair amount of *simplification*, *calculation* and *solving*.

Therefore, powerful simplifiers and other (symbolic or semi-symbolic) algorithms seem to be ideally suited to augment ATPs. In the following we shortly describe two major points of interest in combining SASs with top-down automated theorem provers (here: SETHEO [Let92, GLMS94]).

2 SAS as a Preprocessing Module

Interactive theorem provers and verifiers (e.g., PVS, Isabelle, KIV, HOL) have built-in simplifiers for processing the formulas at hand. As demonstrated by most examples and applications, simplification is one of the most powerful tactics available. Therefore, considerable effort has been spent on developing such simplifiers (cf. e.g., [RSSB98]). As experiments with simplification of proof tasks arising in logic-based component retrieval [FSS98] showed, straight-forward simplification during preprocessing even could solve a large amount of proof tasks without even starting the theorem prover. With standard rewriting and unfolding of quantifiers according to an underlying theory with subsequent simplification, more than 40% of the non-valid proof tasks already could be rewritten to FALSE. More than 25% of the valid conjectures could be detected by this preprocessing module [FSS98].

As shown above, even conservative simplification during preprocessing can lead to dramatic improvement of the prover's capabilities. Therefore, SASs should be used for simplification of complex mathematical (or theory-based) formulas, their solvers should attempt to solve the (often equational) proof obligation (or some parts of it). Furthermore, an SAS could be used to perform partial evaluation whenever possible.

2.1 Dynamic Interfacing

A SAS can also be used during search for the proof. The most naive approach would start the SAS on each subgoal as it occurs and would try to solve it or to contradict it. However, in top-down theorem proving, identical subgoals occur over and over again. Furthermore, most terms in the subgoals are not much instantiated. Thus the simple approach for combining both systems is not feasible. Two more powerful ways for combination come into mind which should be evaluated:

- SETHEO is capable of handling disjunctive (syntactic) equality-constraints, allowing to postpone checking for validity of the constraints. Here, a SAS could simplify the sets of constraints and try to solve them.
- SETHEO can generate unit-lemmata in a bottom-up fashion (Delta-preprocessor [Sch94]). Here filters are extremely important and a combination with a SAS would be most helpful.

3 Important Issues

A combination between an ATP and a SAS is certainly valuable, but various problems have to be addressed:

- technical issues like conversion of *syntax* and *control* of the combined system have to be solved.
- issues of soundness and completeness: Many applications (like logic-based component retrieval) do not essentially require a 100% level of soundness. Because the results are inspected afterwards (e.g., by the user looking at the retrieved component), *some few* errors are acceptable. However, the question remains, inhowfar soundness of a SAS can be estimated such that a reasonable sound system can be designed[1].

[1]Certainly, checking of the behavior of the SAS would be the best way to ensure that

References

[FSS98] Bernd Fischer, Johann Schumann, and Gregor Snelting. Deduction based component retrieval. In W. Bibel and P. Schmitt, editors, *Automated Deduction. A basis for applications*, chapter III.12, pages 265–292. Kluwer, 1998.

[GLMS94] Chr. Goller, R. Letz, K. Mayr, and J. Schumann. SETHEO V3.2: Recent Developments (System Abstract) . In *Proc. CADE 12*, pages 778–782, June 1994.

[Let92] Letz, R. et al. SETHEO: A High-Performance Theorem Prover. *JAR*, 8(2):183–212, 1992.

[RSSB98] W. Reif, G. Schellhorn, K. Stenzel, and M. Balser. *Structured Specifications and Interactive Proofs with KIV*, volume II, chapter II.1, pages 13–40. 1998.

[Sch94] J. Schumann. DELTA — A Bottom-up Preprocessor for Top-Down Theorem Provers, System Abstract. In *CADE 12*, 1994.

[Wan60] H. Wang. Proving theorems by pattern recognition. *CACM*, 4(3):229–243, 1960.

all steps, made by the system are correct. However, the information about which rules or built-in functions have been used to obtain the result is usually not available from the system. However, this is more a copyright problem than a technical one. For example, with the old ('82) version of MuMath, it was possible (with a little bit of assembler hacking) to exactly trace which simplification rules had been applied. Checking such a rule or at least certain preconditions (e.g., don't divide by zero) should be possible within current TP techniques.